CS506 Programming for Computing

HOS03 Functions, Debugging & Exceptions

10/18/2020 Developed by Kim Nguyen

3/28/2022 Revised by Ken Ling

07/06/2023 Reviewed by Maram Elwakeel

10/04/2023 Reviewed by Maram Elwakeel

School of Technology & Computing (STC) @ City University of Seattle (CityU)

Before You Start

- Version numbers may not match with the most current version at the time of writing. If given the option to choose between stable release (long-term support) or most recent, please select the stable release rather than the beta-testing version.
- There might be subtle discrepancies along with the steps. Please use your best judgment while going through this cookbook-style tutorial to complete each step.
- For your working directory, use your course number. This tutorial may use a different course number as an example.
- All the steps and concepts in this tutorial are from the textbook, so if you encounter problems in this tutorial, please try to read and compare the textbook to solve the problem. If you still can't solve the problem, please feel free to contact your course TA.
- Avoid copy-pasting code from the book or the GitHub repository. Instead, type out the code yourself. Resort to copy-pasting only when you are stuck and find that things are not working as expected.

Learning Outcomes

- Learn ways to pass information to functions

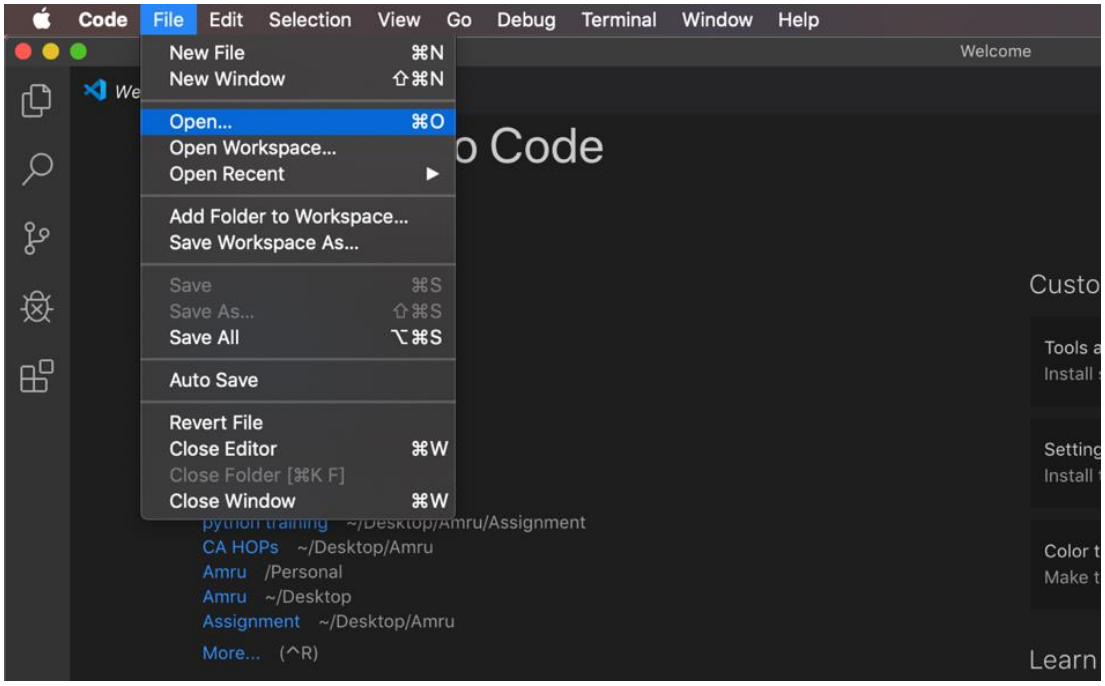
- How to write certain functions whose primary job is to display information and other functions designed to process data and return a value or set of values.
- Learn to store functions in separate files called modules to help organize your main program files.
- How to use debugger tools

Resources

- Python crash course: a hands-on, project-based introduction to programming: Matthes, E. (2019): Available online link
- Functions and Variable Names in Python: https://peps.python.org/pep-0008/#function-and-variable-names
- Python debugger VSCode: https://code.visualstudio.com/docs/python/debugging
- Microsoft debugging Python: https://docs.microsoft.com/en-us/visualstudio/debugger/debugging-absolute-beginners?view=vs-2022&tabs=csharp

Section 1 - Preparation

1) In Visual Studio Code, open the private repository generated when you accepted the HOS03 assignment (If you cannot find that repository in your machine, you might have not cloned the repo, if so, please do before proceeding).



2) Open the terminal from the VSCode by hitting the control + ~ key, navigate into Module 3 folder using the following command:

3) `cd Module 3`

Section 2 - Python Functions

1) A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

2) A function can be written to as a set of statements that take inputs and perform specific computation and produces output.

3) **Syntax:**

    i. def function_name(parameters):

    ii. code

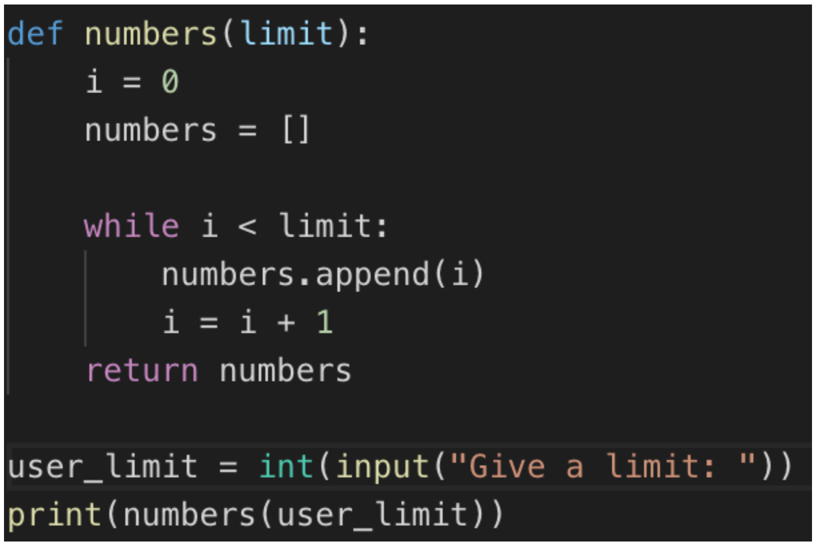
    iii. return [expression]

    - **Note:** The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None. It is not mandatory to use a return statement.

4) Naming conversions are extremely important for code readability, reusability and consistency. Detailed information about Python naming conventions for functions and variables can be found here.

5) A function can process some data and then return a value or set of values. The value the function returns is called a return value. The return statement takes a value from inside a function and sends it back to the line that called the function. Return values allow you to move much of your program's grunt work into functions, which can simplify the body of your program.

6) Create a file **print_numbers.py** and type the following code.



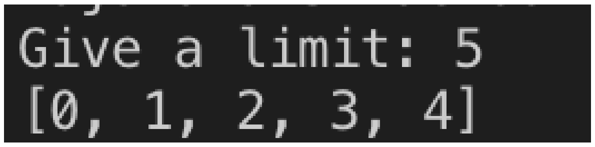
```python
def numbers(limit):
    i = 0
    numbers = []

    while i < limit:
        numbers.append(i)
        i = i + 1
    return numbers

user_limit = int(input("Give a limit: "))
print(numbers(user_limit))
```

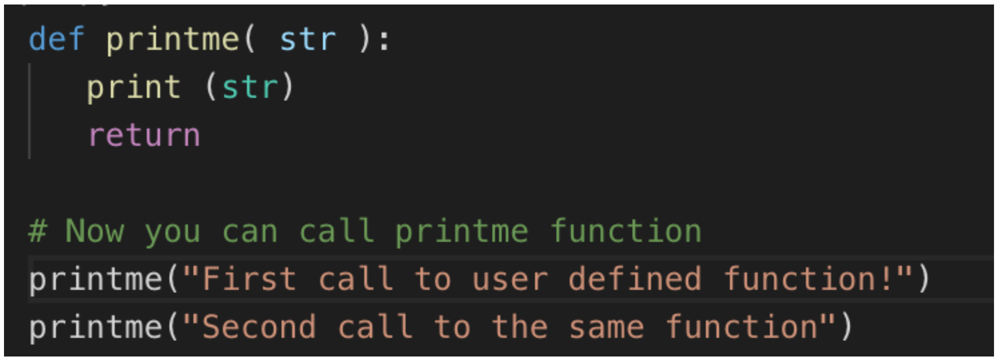
7) **Challenge:** in the above code, remove the "print(numbers(user_limit))" statement. What will the output look like now?

8) In the terminal type **python3 print_numbers.py** to check the output



```
Give a limit: 5
[0, 1, 2, 3, 4]
```

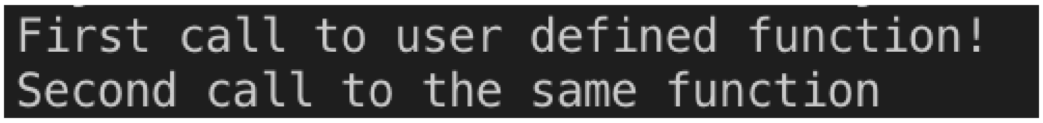
9) A function doesn't always have to return something, sometime, a function just print something out, it is the same as return None. Example as below (You don't have to write the below code):



```python
def printme( str ):
   print (str)
   return

# Now you can call printme function
printme("First call to user defined function!")
printme("Second call to the same function")
```

i. Result:



```
First call to user defined function!
Second call to the same function
```

Section 3 - Passing Arguments

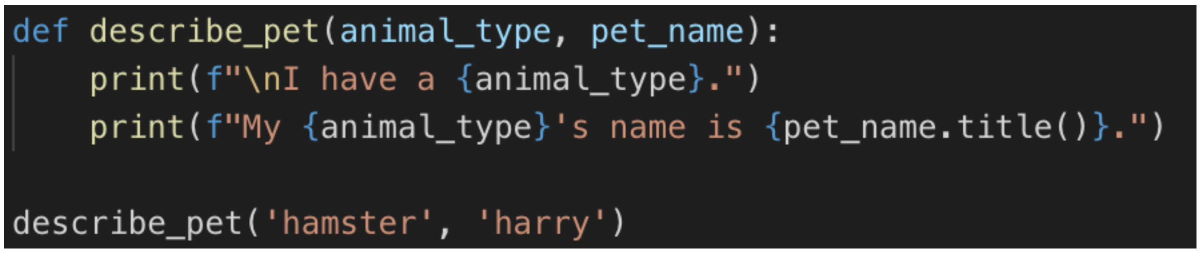
1) Because a function definition can have multiple parameters, a function call may need multiple arguments. You can pass arguments to your functions in several ways.

   i. Positional arguments: which need to be in the same order as the parameters were written

   ii. Keyword arguments: where each argument consists of a variable name and a value

   iii. Arbitrary Number of Arguments: Sometimes you will not know ahead of time how many arguments a function needs to accept. In that case arbitrary can be used.

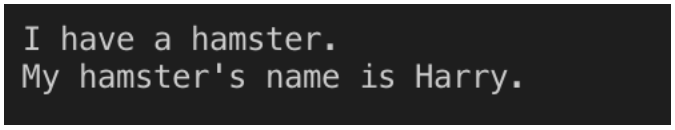
2) Create a file **positional.py** and type the following code.



```python
def describe_pet(animal_type, pet_name):
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet('hamster', 'harry')
```

3) In the terminal type **python3 positional.py** to check the output



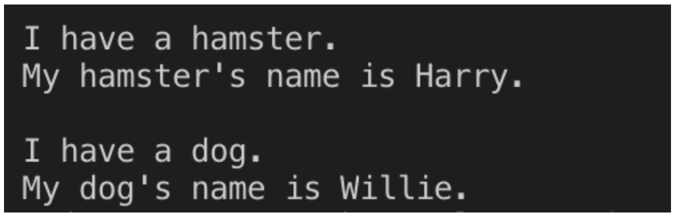
```
I have a hamster.
My hamster's name is Harry.
```

4) The definition shows that this function needs a type of animal and the animal's name. When we call describe_pet(), we need to provide an animal type and a name, in that order. For example, in the function call, the argument 'hamster' is assigned to the parameter animal_type and the argument 'harry' is assigned to the parameter pet_name. In the function body, these two parameters are used to display information about the pet being described.

   i. The order matters in positional arguments. The parameters specified in the function definition have to be in the same order as how the function is called. There won't be any error showing when running the code, but the result might not be as you expected it to be.

5) **Challenge:**

   i. Add to the code above to receive the following result:



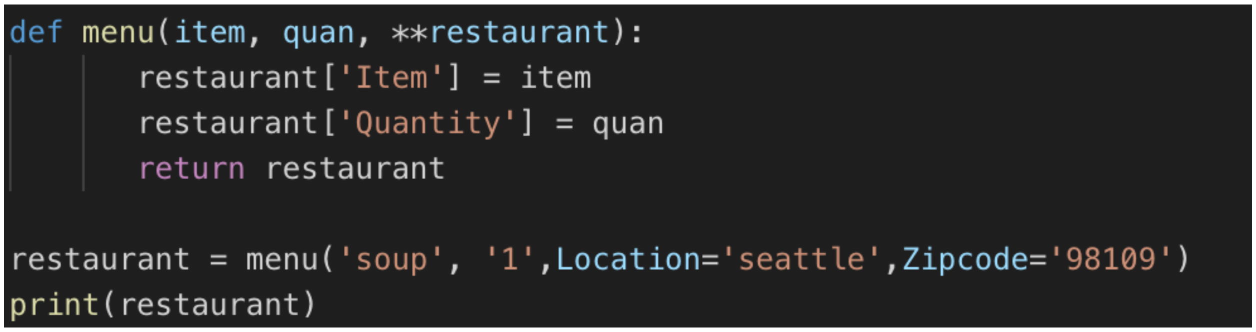
```
I have a hamster.
My hamster's name is Harry.

I have a dog.
My dog's name is Willie.
```

Section 4 - Arbitrary number of arguments

1) Sometimes you'll want to accept an arbitrary number of arguments, but you won't know ahead of time what kind of information will be passed to the function. In this case, you can write functions that accept as many key-value pairs as the calling statement provides.

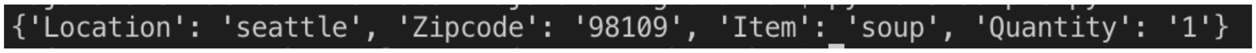
2) Create a file **arbitrary_dict.py** and type the following code



```
def menu(item, quan, **restaurant):
        restaurant['Item'] = item
        restaurant['Quantity'] = quan
        return restaurant

restaurant = menu('soup', '1',Location='seattle',Zipcode='98109')
print(restaurant)
```

3) In the terminal type **python3 arbitrary_dict.py** to check the output



```
{'Location': 'seattle', 'Zipcode': '98109', 'Item': 'soup', 'Quantity': '1'}
```

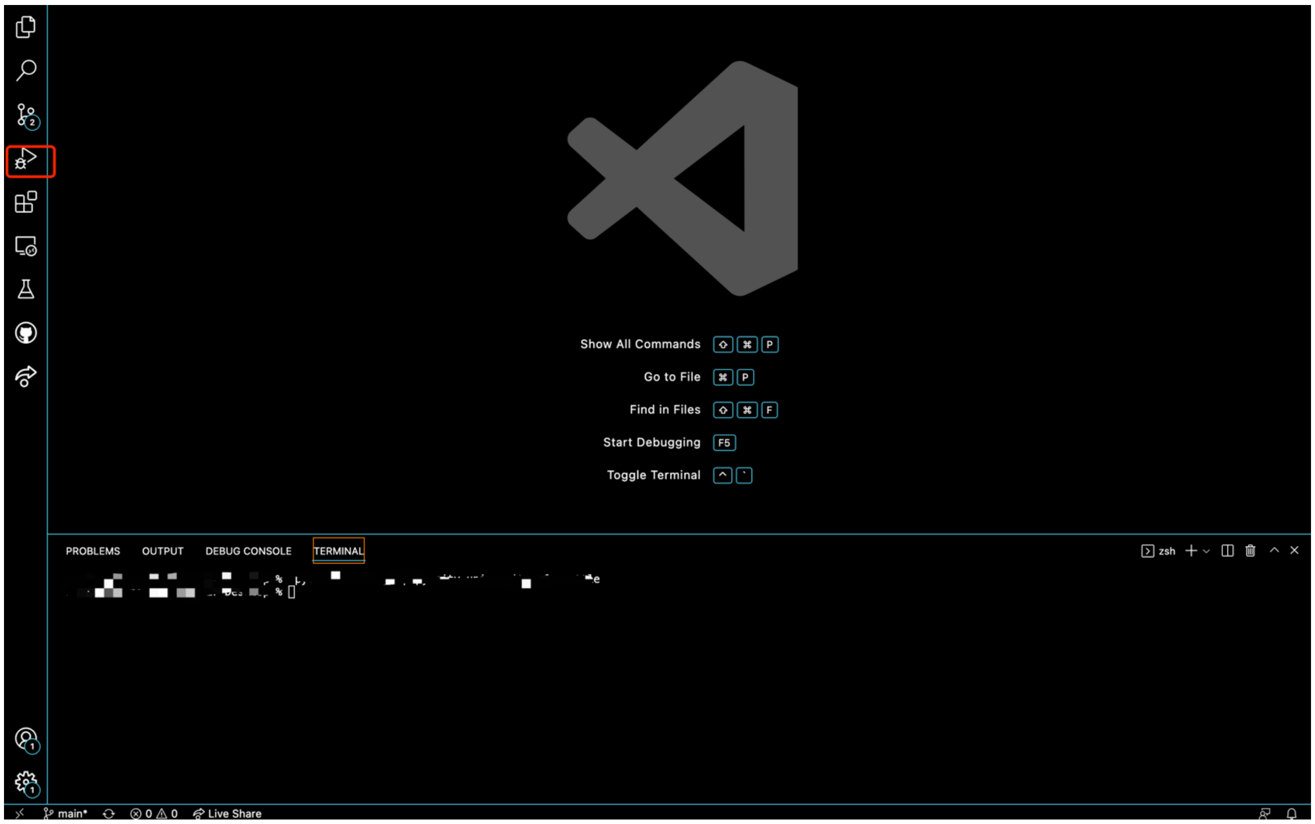
4) The “menu” is the definition where user can pass the item and quantity, then it allows the user to pass in as many name-value pairs as they want. The double asterisks before the parameter **restaurant cause Python to create an empty dictionary called “restaurant” and pack whatever name-value pairs it receives into this dictionary.

Section 5 - Python Debugging

1) Debugging is important when you are stuck with errors including syntax errors.

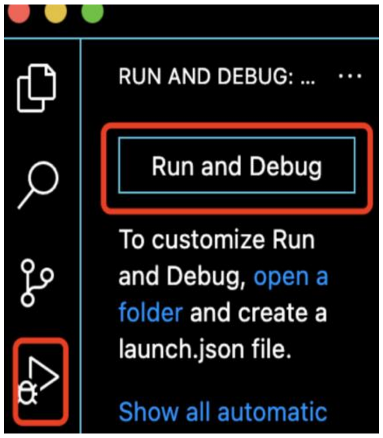
2) For more information about debugging click here.

3) Visual Studio Code has a built-in debugger that can be used by clicking on the debugger icon as shown in the picture below:



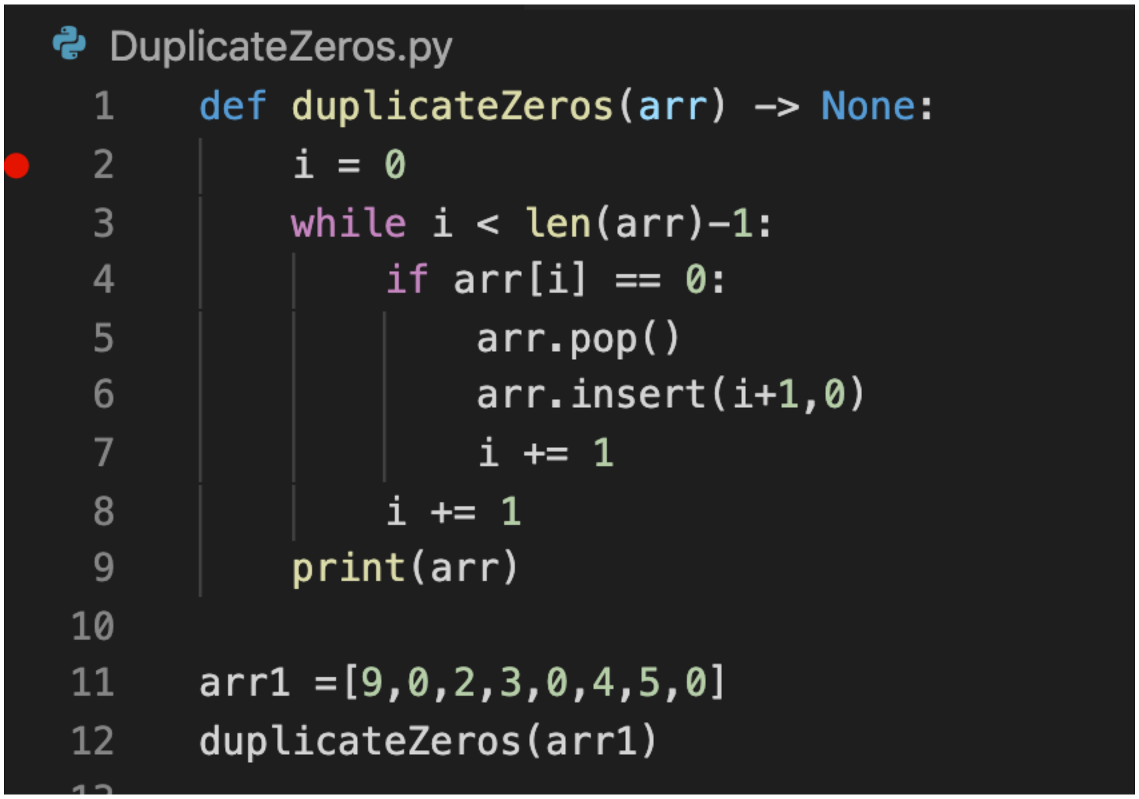
[redacted]

4) To initialize the debug configurations, first select the Run and Bug button in the sidebar.



To customize Run and Debug, open a folder and create a launch.json file.

5) The Command Palette will offer a configuration menu where you can select the type of debugging setup you want for the current file. For the time being, choose Python File from the Select a debug configuration selection.
6) Find more information about python debugging in Visual Studio Code here.
7) Create **DuplicateZeros.py** file, and type the following code:



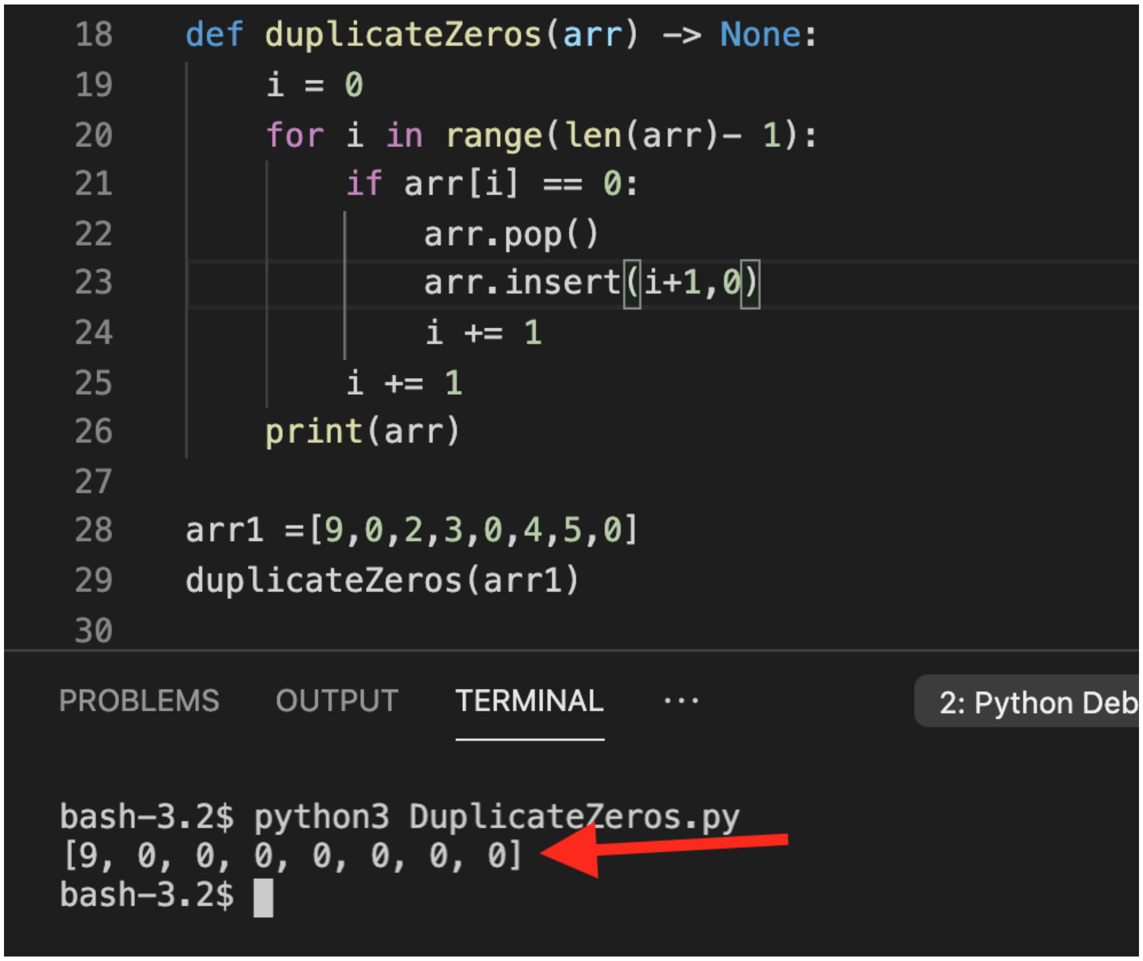
DuplicateZeros.py

```
def duplicateZeros(arr) -> None:
    i = 0
    while i < len(arr)-1:
        if arr[i] == 0:
            arr.pop()
            arr.insert(i+1,0)
            i += 1
        i += 1
    print(arr)

arr1 =[9,0,2,3,0,4,5,0]
duplicateZeros(arr1)
```

8) Watch the following video to see how we can debug this program using Debugger tool:
9) https://youtu.be/wODNZqYU8fU

10) Let’s change the while loop to a for loop and see what happens:



```python
def duplicateZeros(arr) -> None:
    i = 0
    for i in range(len(arr)- 1):
        if arr[i] == 0:
            arr.pop()
            arr.insert(i+1,0)
            i += 1
        i += 1
    print(arr)

arr1 =[9,0,2,3,0,4,5,0]
duplicateZeros(arr1)
```

```
bash-3.2$ python3 DuplicateZeros.py
[9, 0, 0, 0, 0, 0, 0, 0]
bash-3.2$
```

11) Clearly, the result is totally different.

12) **Challenge:**

i. **Explain why using for loop in the above program does not work.**

ii. ***Hint: Use Debugger Tool***

Push your work to GitHub

Push your work to GitHub Follow the instructions on the CityU STC TA Center Github.io Submit your work page.